

# Chapter 7 - Data Modelling

SI539 - Charles Severance

Textbook: Build Your own Ruby on Rails Application by Patrick Lenz (ISBN:978-0-975-8419-5-2)  
See also: Chapter 18 Agile Web Development with Rails ISBN: 0-9776166-3-0

# Basic Idea

- Simple - One table one Object
- But often objects are related to each other
- A Site has a User who created the site and as such has more permissions than other users

# Database Design Light

- We will do “design light” - focus on the business objects
- We will focus on clearly expressing relationships
- We will only do a little bit about performance

# The Basics

```
class CreateMembers < ActiveRecord::Migration
  def self.up
    create_table :members do |t|
      t.column :name, :string
      t.column :email, :string
    end
  end
end
```

# Two Migrations

```
class CreateChats < ActiveRecord::Migration
  def self.up
    create_table :chats do |t|
      t.column :chatmsg, :string
      t.column :member_id, :integer
      t.column :created_at, :datetime
    end
  end
end
```

SQLite Database Browser - /Users/csev/Desktop/teach/a539/w...

The screenshot shows the SQLite Database Browser interface. The title bar reads "SQLite Database Browser - /Users/csev/Desktop/teach/a539/w...". The toolbar contains icons for file operations, database management, and help. Below the toolbar, there are three tabs: "Database Structure" (selected), "Browse Data", and "Execute SQL". The main window displays the database schema. A table lists objects by name, type, and schema. The "chats" table has fields: id (INTEGER PRIMARY KEY), chatmsg (varchar(255)), member\_id (integer), and created\_at (datetime). The "members" table has fields: id (INTEGER PRIMARY KEY), name (varchar(255)), and email (varchar(255)). A third table, "schema\_info", is also listed. The schema for the "chats" table is shown as a CREATE TABLE statement.

Name	Object	Type	Schema
▼chats	table		CREATE TABLE chats ("id" IN...
id	field	INTEGER PRIMARY KEY	
chatmsg	field	varchar(255)	
member_id	field	integer	
created_at	field	datetime	
▼members	table		CREATE TABLE members ("i...
id	field	INTEGER PRIMARY KEY	
name	field	varchar(255)	
email	field	varchar(255)	
►schema_info	table		CREATE TABLE schema_info ...

# Two Tables

# Two Models

```
class Member < ActiveRecord::Base  
end
```

Why are these files so small?

```
class Chat < ActiveRecord::Base  
  belongs_to :member  
end
```

# What Does ActiveRecord Do?

- When a model which extends ActiveRecord is created, it looks for a table which is the same as the name of the model - but pluralised
  - Member -> members
  - Chat -> chats
- Then it looks at the names of the columns in the table....

# Columns => Methods

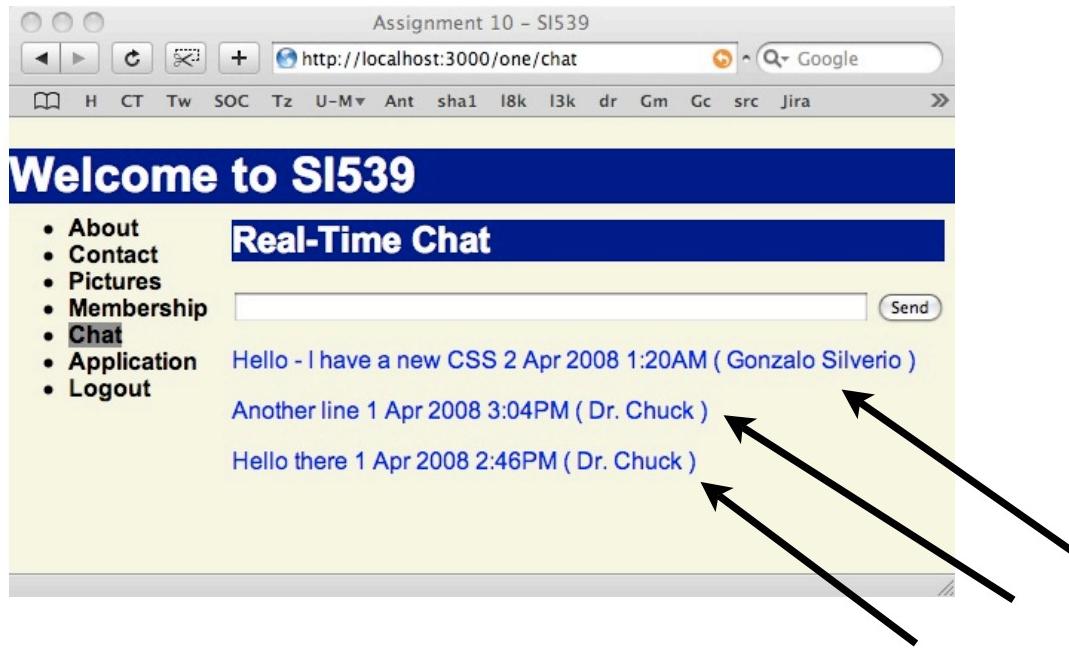
```
zzz = Member.new()
zzz.name = "Chuck"
zzz.email = "csev@umich.edu"
zzz.save
logger.info zzz.id

abc = Member.find_by_name("Chuck")
def = Member.find_by_email("csev@umich.edu")
xyz = Member.find_by_name_and_email(
    "Chuck", "csev@umich.edu")
```

```
class Member < ActiveRecord::Base
end
```

Name	Object	Type
chats	table	
id	field	INTEGER PRIMARY KEY
chatmsg	field	varchar(255)
member_id	field	integer
created_at	field	datetime
members	table	
id	field	INTEGER PRIMARY KEY
name	field	varchar(255)
email	field	varchar(255)
schema_info	table	

# Relationships...



We want to keep track of who is the “owner” of each chat message...  
Who does this chat message “belong to”???

# Keeping Track

- We could store the name as a string in each and every record in the chats table
- This would be \*BAD\*
- If someone changed their name it would be scattered \*everywhere\*

```
class CreateChats < ActiveRecord::Migration
  def self.up
    create_table :chats do |t|
      t.column :chatmsg, :string
      t.column :membername :string
      t.column :created_at, :datetime
    end
  end
end
```

# Database Nomalization (3NF)

- There is \*tons\* of database theory - way too much to understand without excessive predicate calculus
- Short form
  - Do not replicate data - reference data
  - Use integers for keys and for references

[http://en.wikipedia.org/wiki/Database\\_normalization](http://en.wikipedia.org/wiki/Database_normalization)

# Better Reference Pattern

We use integers to reference rows in another table.

Table: members			
	id	name	email
1	1	Dr. Chuck	csev@umich.edu
2	2	Gonzalo Silverio	gsilver@umich.edu

Table: chats				
	id	chatmsg	member_id	created_at
1	1	Hello there	1	2008-04-01 14:45:00
2	2	Another line	1	2008-04-01 14:45:00
3	3	Hello - I have a	2	2008-04-02 01:00:00

# Rails Automates This Pattern

```
class CreateChats < ActiveRecord::Migration
  def self.up
    create_table :chats do |t|
      t.column :chatmsg, :string
      t.column :member_id, :integer
      t.column :created_at, :datetime
    end
  end
end
```

In the model we indicate a relationship..

In the database we add an integer column to store the reference.

```
class Chat < ActiveRecord::Base
  belongs_to :member
end
```

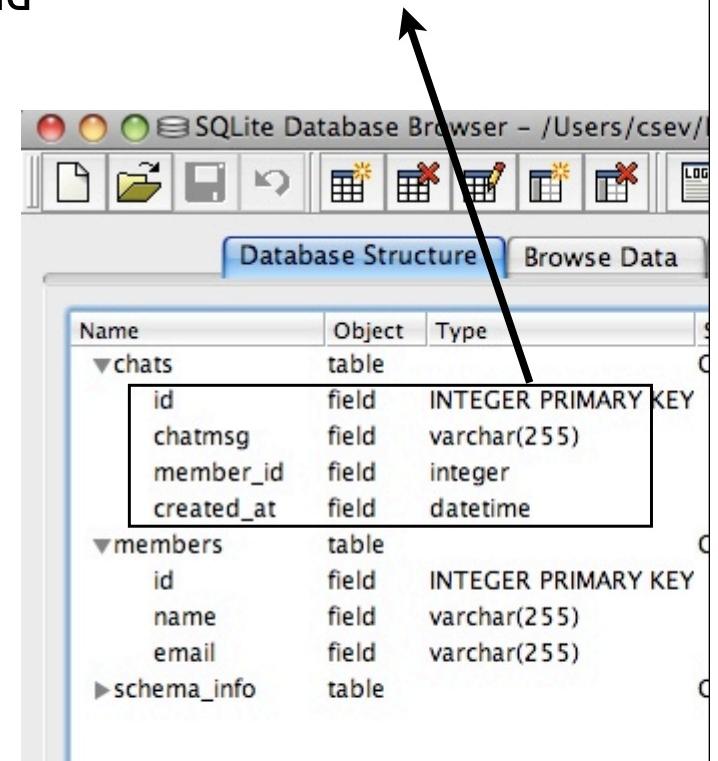
# More Methods...

```
abc = Member.find_by_name("Chuck")
```

```
ch = Chat.new()
ch.chatmsg = "hi"
ch.member = abc
ch.save
```

The member method in the Chat object takes a Member object. All the connections happen.

```
class Chat < ActiveRecord::Base
  belongs_to :member
end
```



Name	Object	Type
chats	table	
id	field	INTEGER PRIMARY KEY
chatmsg	field	varchar(255)
member_id	field	integer
created_at	field	datetime
members	table	
id	field	INTEGER PRIMARY KEY
name	field	varchar(255)
email	field	varchar(255)
schema_info	table	

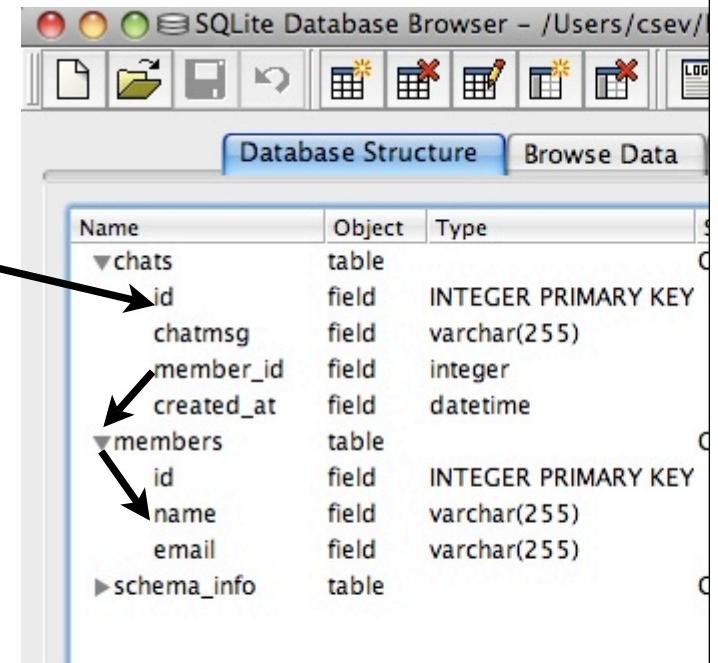
# Methods upon methods!

```
lmn = Chat.find(2)
```

```
logger.info lmн.member.name
```

The member method in the Chat object takes a Member object. All the connections happen.

```
class Chat < ActiveRecord::Base  
  belongs_to :member  
end
```



A screenshot of the SQLite Database Browser application. The title bar reads "SQLite Database Browser - /Users/csev/". The main window shows a table of database schema. The columns are "Name", "Object", and "Type". There are two tables listed: "chats" and "members". The "chats" table has fields: id (INTEGER PRIMARY KEY), chatmsg (varchar(255)), member\_id (integer), and created\_at (datetime). The "members" table has fields: id (INTEGER PRIMARY KEY), name (varchar(255)), and email (varchar(255)). A "schema\_info" table is also listed. Two arrows point from the text "The member method in the Chat object takes a Member object. All the connections happen." to the "members" table in the database browser.

Name	Object	Type
chats	table	
id	field	INTEGER PRIMARY KEY
chatmsg	field	varchar(255)
member_id	field	integer
created_at	field	datetime
members	table	
id	field	INTEGER PRIMARY KEY
name	field	varchar(255)
email	field	varchar(255)
schema_info	table	

```
<% for chat in @chats %>
  <p><%= chat.chatmsg %>
<%= chat.created_at %>
<% if chat.member != nil %>
  ( <%= chat.member.name %> )
<% end %>
<% end %>
```

```
Hello - I have a new CSS 2 Apr 2008 1:20AM ( Gonzalo Silverio )
Another line 1 Apr 2008 3:04PM ( Dr. Chuck )
Hello there 1 Apr 2008 2:46PM ( Dr. Chuck )
```

```
Processing OneController#chatcontent (for 127.0.0.1 at 2008-04-02 01:32:54) [POST]
Session ID: bc33c1894efcd1d819898de850277129
Parameters: {"action"=>"chatcontent", "controller"=>"one"}
Chat Load (0.001043)   SELECT * FROM chats ORDER BY chats.created_at DESC LIMIT 5
We found 3 chats
Rendering one/chatcontent
Member Load (0.000330)   SELECT * FROM members WHERE (members."id" = 2)
Member Load (0.000284)   SELECT * FROM members WHERE (members."id" = 1)
Member Load (0.000276)   SELECT * FROM members WHERE (members."id" = 1)
```

**Real Databases are More  
Complex...**

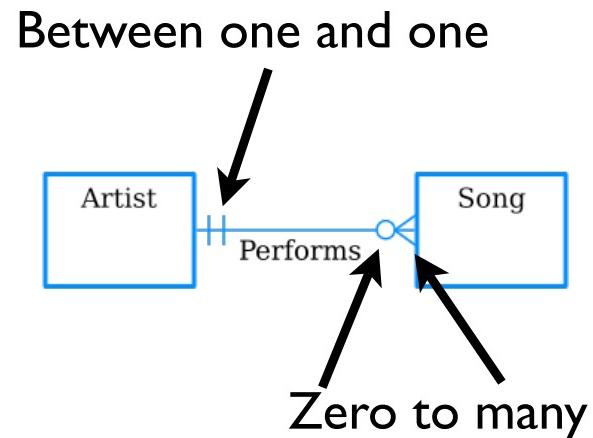
# Database Design

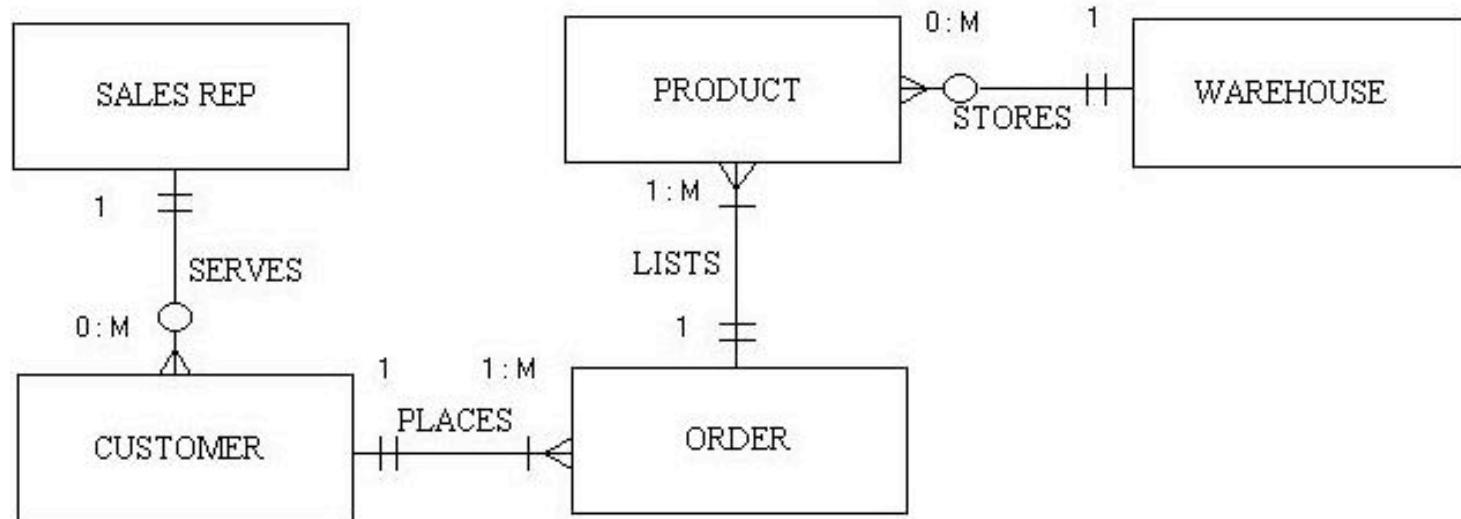
- Database design is an art form of its own with particular skills and experience
- Our goal is to avoid the really bad mistakes and design clean and easily understood databases
- Others may performance tune things later
- Database design starts with a picture...

# Entity - Relationship - Diagram

- Vertical line “one”
- Circle “zero”
- Crow foot “many”
- Two marks give a range
- This is a “one-to-many relationship between Artist and song”
- An artist has zero or more songs..

[http://en.wikipedia.org/wiki/Entity-relationship\\_model](http://en.wikipedia.org/wiki/Entity-relationship_model)





**Figure 1. Entity-Relationship Diagram**

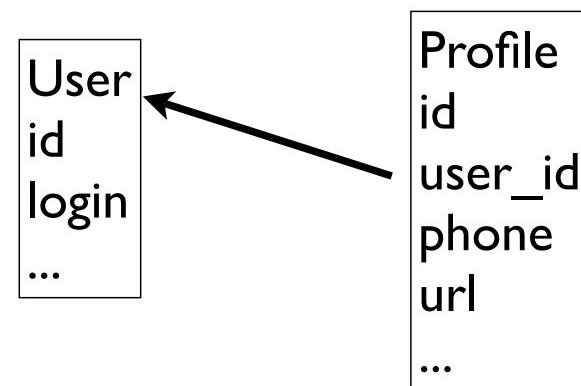
- \* 1 INSTANCE OF A SALES REP SERVES 1 TO MANY CUSTOMERS
- \* 1 INSTANCE OF A CUSTOMER PLACES 1 TO MANY ORDERS
- \* 1 INSTANCE OF AN ORDER LISTS 1 TO MANY PRODUCTS
- \* 1 INSTANCE OF A WAREHOUSE STORES 0 TO MANY PRODUCTS

# The Basic Relationships

- One to zero or one
- One to many
- Many to many

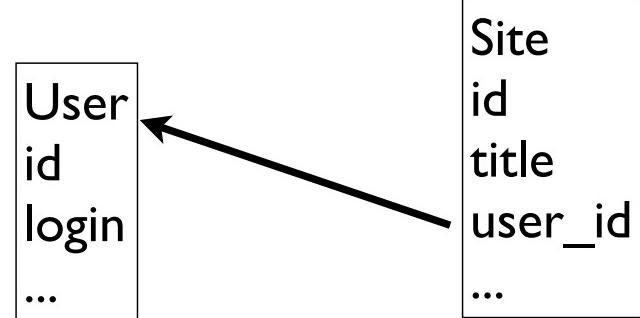
# One to One

- Each User object may have a profile object
- But a User never has more than one profile
- “One to Zero or One”



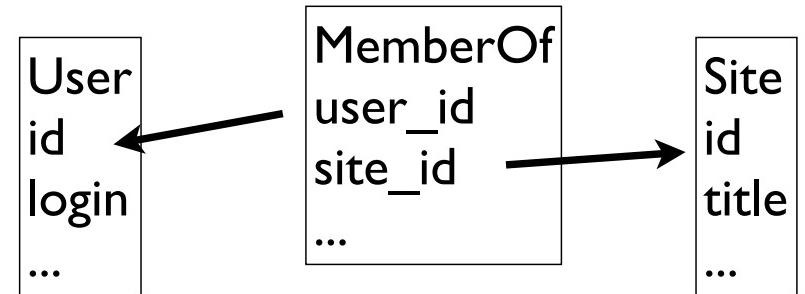
# One to Many

- Each User object may have many sites
- A Site always has one single “owner”
- A User may also have zero sites



# Many to Many

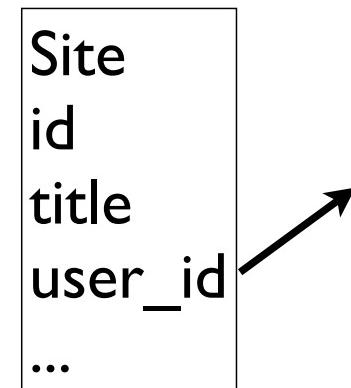
- A User can be a member of many Sites
- A Site can have many members
- We cannot put this in a column of either table
- We need a separate “relates” table - or join table



These relates tables often do not have a separate primary key - the two fields form a composite primary key.

# Three Kinds of Keys

- Primary key - generally an integer auto-increment field
- Logical key - What the outside world uses for lookup
- Foreign key - generally an integer key point to a row in another table



# Primary Key Rules

- Rails encourages you to follow best practices
- Never use your logical key as the primary key
- Logical keys can and do change albeit slowly
- Relationships that are based on matching string fields are far less efficient than integers performance-wise

```
User  
id  
login  
password  
name  
email  
created_at  
modified_at  
login_at
```

SQLite Database Browser - /Users/csev/dev/toozday/trunk/db/development.sqlite3

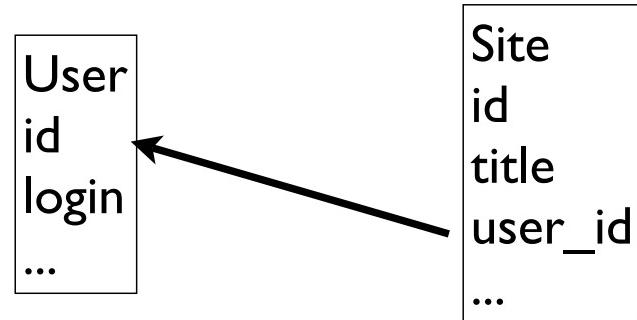
Database Structure    Browse Data    Execute SQL

Name	Object	Type	Schema
►comments	table		CREATE TABLE comments ("id" INTEGER PRIMARY KEY N...
►schema_info	table		CREATE TABLE schema_info (version integer)
►sites	table		CREATE TABLE sites ("id" INTEGER PRIMARY KEY NOT NU...
►sitetypes	table		CREATE TABLE sitetypes ("id" INTEGER PRIMARY KEY NO...
▼users	table		CREATE TABLE users ("id" INTEGER PRIMARY KEY NOT N...
	id	field	INTEGER PRIMARY KEY ←
	login	field	varchar(255)
	password	field	varchar(255)
	name	field	varchar(255)
	email	field	varchar(255)
	created_at	field	datetime
	modified_at	field	datetime
	login_at	field	datetime
►wikis	table		CREATE TABLE wikis ("id" INTEGER PRIMARY KEY NOT N...

Auto-Increment

# Foreign Keys

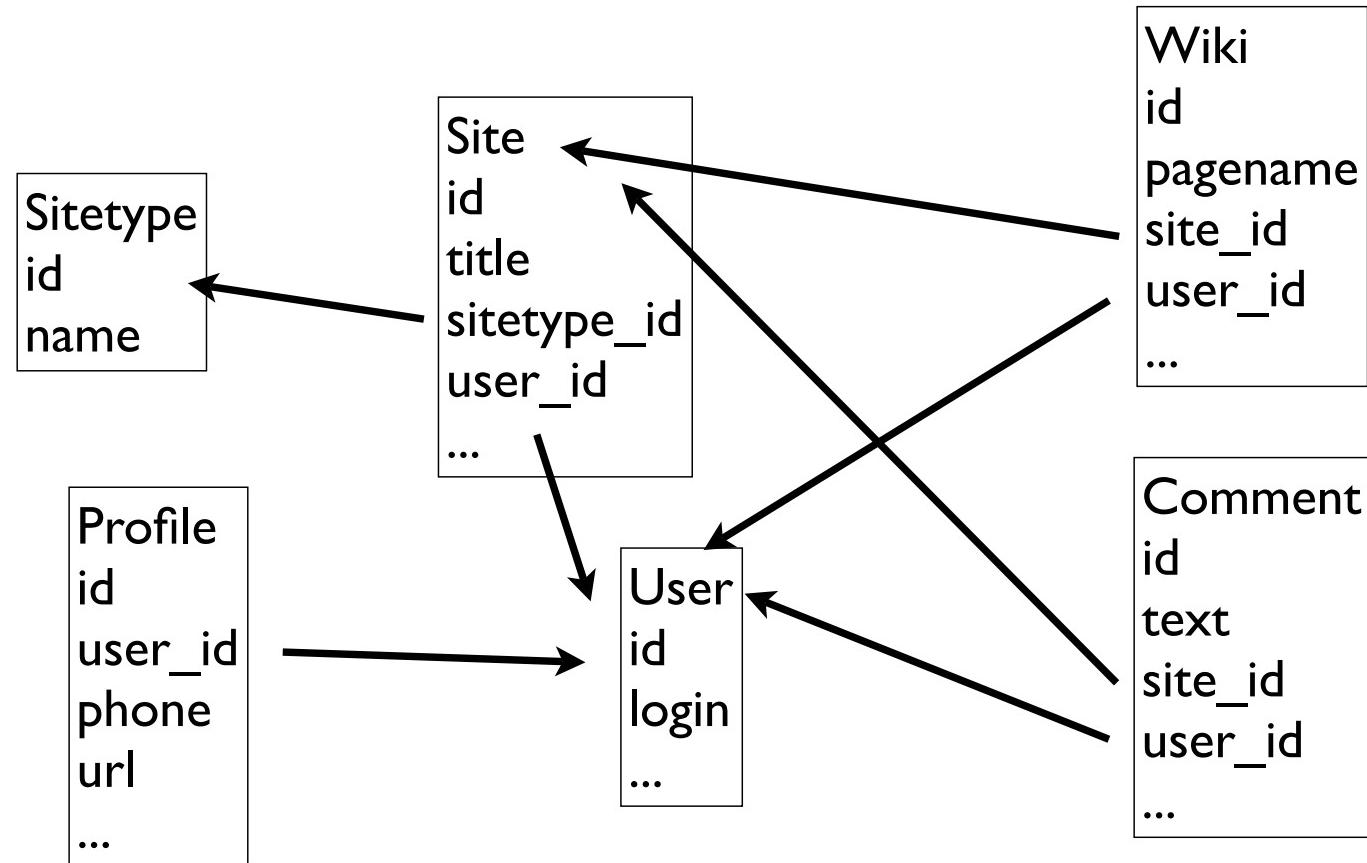
- A foreign key is when a table has a column that contains a key which points the primary key of another table.
- When all primary keys are integers, then all foreign keys are integers - this is good - very good
- And Rails pretty much forces this



# Where to start designing?

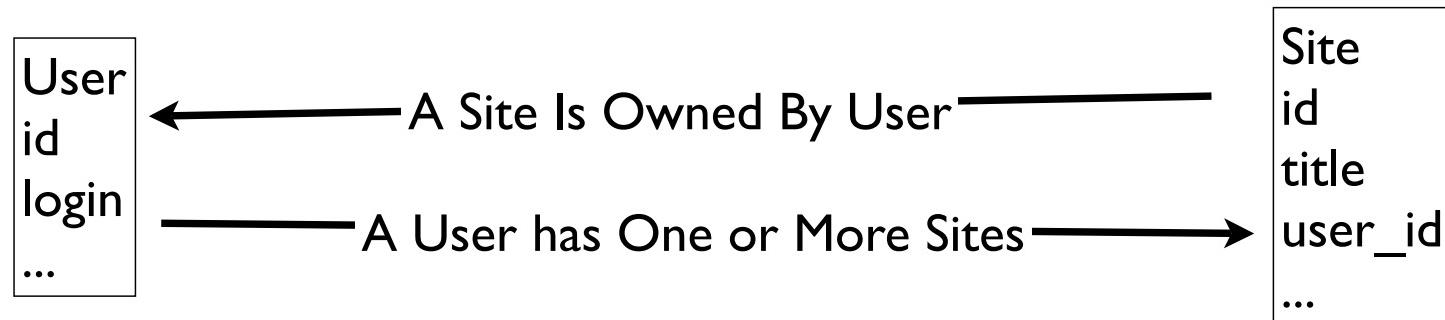
- Generally start with the people / users
- For most applications, people are central elements
- Often “all roads lead to people”
- Draw the “User” first on your whiteboard and then build out from the user

```
User
id
login
password
name
email
created_at
modified_at
login_at
```



# Relationships

- These relationships are “connections” - they connect two things together
- “Relational” databases - what related to what - and what is the nature of the relationships



# In Rails

- Database designers things of relationships as outside or between of the objects
- Rails moves the relationships into the objects
- We define primary, logical, and foreign keys in the migrations
- We define the relationships in the models

# 003\_create\_sites.rb

```
create_table :sites do |t|
  # Rails adds “id” for us D.R.Y.
  t.column :title, :string
  t.column :description, :string
  t.column :sitetype, :string
  t.column :sitetype_id, :integer
  t.column :user_id, :integer
  t.column :created_at, :datetime
end
```

Foreign keys end in `_id`  
Rails knows these are  
not human readable  
nor settable values.

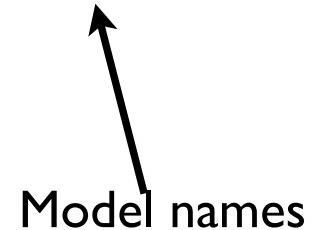
# Sites Model (sites.rb)

```
class Site < ActiveRecord::Base
  belongs_to :user
  belongs_to :sitetype
end
```

# Sites Model (sites.rb)

- We indicate our “outward” relationships from this model to the other models
- There are several kinds of relationships

```
class Site < ActiveRecord::Base  
  belongs_to :user  
  belongs_to :sitetype  
end
```



Model names

# Rails Model Relationships

- `belongs_to` - This model belongs to or is owned by some other model
- `has_one` - This model is related to exactly one instance of another model - “joined at the hip”
- `has_many` - This model is possibly related to many instances of another model
- `has_and_belongs_to_many`

```
class Profile  
  belongs_to :user  
end
```

```
class User  
  has_many :comments  
  has_many :sites  
  has_many :wikis  
  has_one :profile  
end
```

```
class Comment  
  belongs_to :user  
  belongs_to :site  
end
```

```
class Sitetype  
  has_many :sites  
end
```

```
class Site  
  belongs_to :user  
  belongs_to :sitetype  
end
```

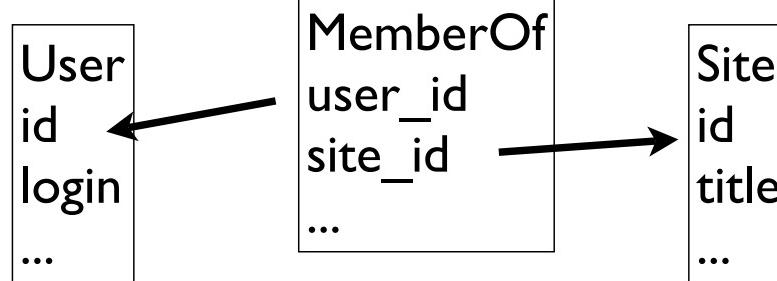
```
class Wiki  
  belongs_to :user  
  belongs_to :site  
end
```

```
class User  
  has_many :memberships  
end
```

```
class Site  
  has_many: memberships  
end
```

## Many to Many in Objects

```
class Membership  
  belongs_to :user  
  belongs_to :site  
end
```



# Programming belongs\_to

```
class Comment
  belongs_to :user
  belongs_to :site
end
```

We add methods to the comment object to set and get the related user object.

```
u = User.new
u.login= "csev"
u.save

c = Comment.new
c.user = u
c.text = "Cool stuff"
c.save

c = Comment.find(2)
logger.info c.user.name
```

# Using Conditions

```
class Comment
  belongs_to :user
  belongs_to :site
end
```

Note: Conditions expressed in terms of table names and table field names - not model names.

```
@comms = Comment.find(:all,
  :conditions => [ "site_id = ?", 4])
```

Only Retrieve those comments which have a site\_id of 4.

# Giving Hints

```
class Comment
  belongs_to :user
  belongs_to :site
end
```

While you are talking to the database, go ahead and pre-retrieve the user objects that each comment belongs to because I will be using them all later.

```
@comms = Comment.find(:all,
  :conditions => [ "site_id = ?", 4],
  :include => :user)
```

```
<% for comment in @comms %>
  By <%= comment.user.login %>
<% end %>
```

```
@comms = Comment.find(:all,  
:conditions => [ "site_id = ?", 4])
```

```
<% for comment in @comms %>  
  By <%= comment.user.login %>  
<% end %>
```

```
SELECT * FROM comments WHERE (site_id = '4')  
Rendering comments/ajaxstart  
SELECT * FROM users WHERE (users."id" = 3)  
SELECT * FROM users WHERE (users."id" = 3)  
SELECT * FROM users WHERE (users."id" = 3)  
SELECT * FROM users WHERE (users."id" = 1)
```

```
@comms = Comment.find(:all,  
:conditions => [ "site_id = ?", 4],  
:include => :user)
```

```
<% for comment in @comms %>  
  By <%= comment.user.login %>  
<% end %>
```

```
SELECT comments."id" AS t0_r0,  
comments."text" AS t0_r1, comments."user_id"  
      AS t0_r2, comments."site_id" AS t0_r3,  
comments."created_at" AS t0_r4, users."id" AS  
t1_r0, users."login" AS t1_r1, users."password"  
AS t1_r2, users."name" AS t1_r3, users."email"  
      AS t1_r4, users."created_at" AS t1_r5,  
users."modified_at" AS t1_r6, users."login_at"  
AS t1_r7 FROM comments LEFT OUTER JOIN  
users ON users.id = comments.user_id  
      WHERE (site_id = '1')  
Rendering comments/ajaxstart
```

# SQL

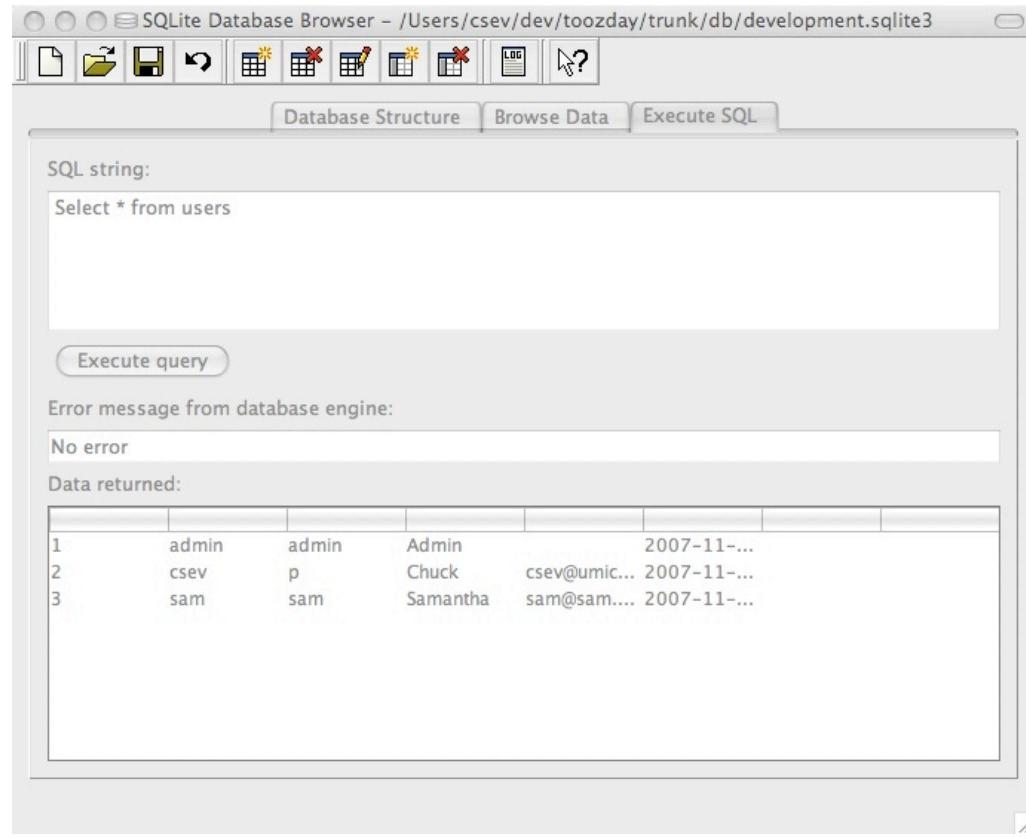
- The language we use to talk to databases
  - `SELECT * FROM users WHERE login = 'csev'`
- Elegant language and very powerful
- Portable subset for common operations but not portable outside standard area
- Very non-portable performance tweaks

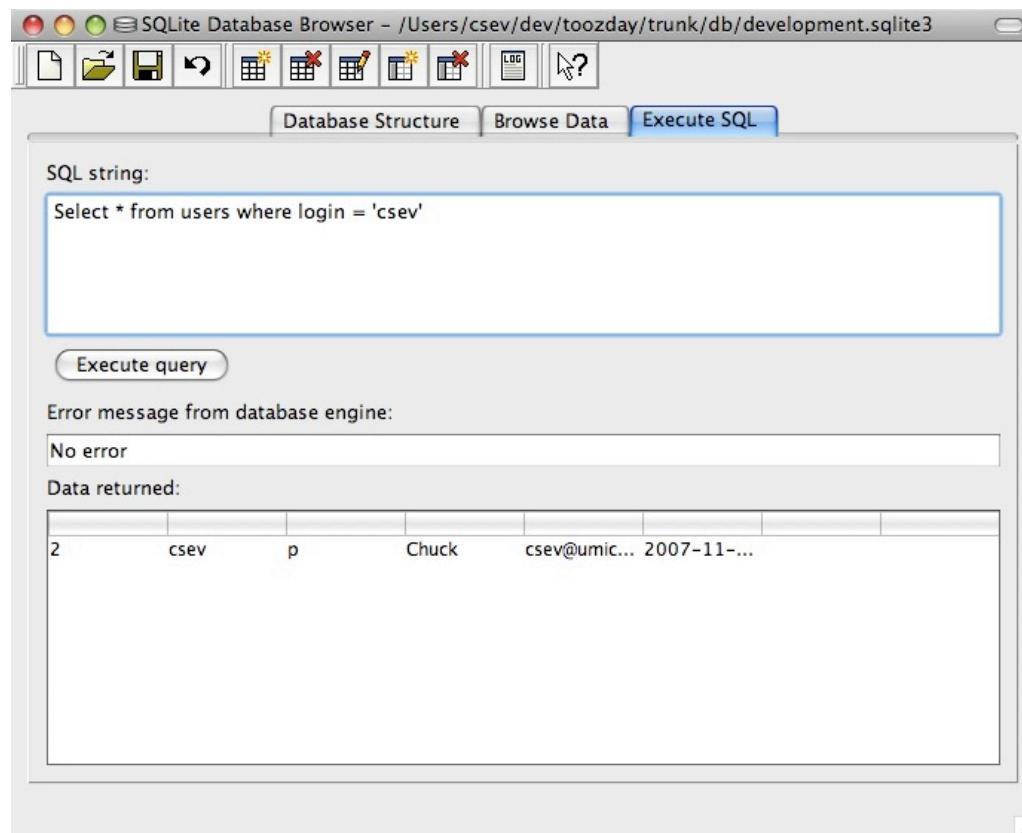
# SQL C.R.U.D.

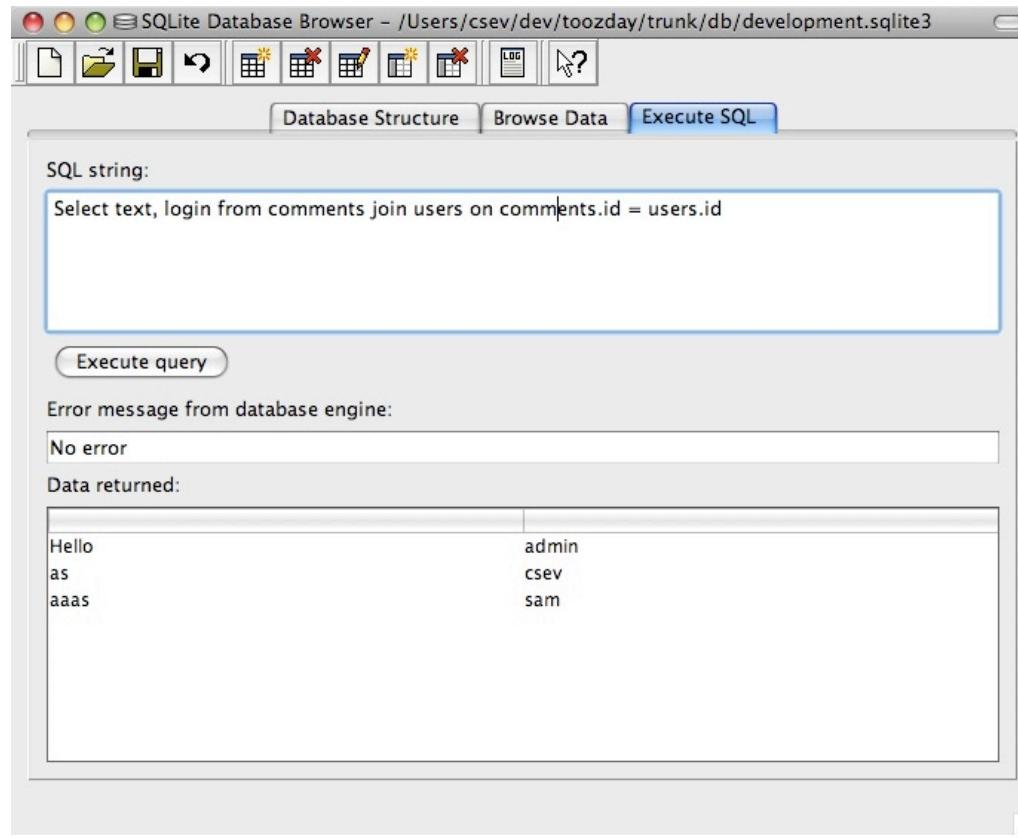
- INSERT - Create
- SELECT - Read
- UPDATE - Update
- DELETE - Delete

# SQL Clauses

- WHERE clause - which records to apply operation to
- ORDER BY clause - what order do the records come back
- JOIN clause - Pull related data from multiple tables







```
SELECT * FROM comments WHERE (site_id = '4')
```

```
SELECT * FROM users WHERE (users."id" = 3)
```

```
SELECT * FROM users WHERE (users."id" = 1)
```

```
SELECT comments."id" AS t0_r0, comments."text" AS t0_r1,  
comments."user_id" AS t0_r2, comments."site_id" AS t0_r3,  
comments."created_at" AS t0_r4, users."id" AS t1_r0, users."login" AS  
t1_r1, users."password" AS t1_r2, users."name" AS t1_r3, users."email" AS  
t1_r4, users."created_at" AS t1_r5, users."modified_at" AS t1_r6,  
users."login_at" AS t1_r7 FROM comments LEFT OUTER JOIN users ON  
users.id = comments.user_id WHERE (site_id = '1')
```

# Summary

- There are many complex elements here - the concepts are simple but things get more complex as applications grow
- Good book: Agile Web Development with Rails Chapter 18
- Advanced topics
  - Has and belongs to many
  - Delete policy - Cascade, etc..